Distributed QoS Guarantees for Realtime Traffic in Ad Hoc Networks

Yaling Yang
Department of Computer Science
University of Illinois at Urbana-Champaign
Email: yyang8@uiuc.edu

Robin Kravets
Department of Computer Science
University of Illinois at Urbana-Champaign
Email: rhk@cs.uiuc.edu

***Abstract*— In this paper, we propose a new cross-layer framework, named QPART (QoS Protocol for Adhoc Realtime Traffic), which provides QoS guarantees to realtime multimedia applications for wireless ad hoc networks. By adapting the contention window sizes at the MAC layer, QPART schedules packets of flows according to their unique QoS requirements. QPART implements priority-based admission control and conflict resolution to ensure that the requirements of admitted realtime flows is smaller than the network capacity. The novelty of QPART is that it is robust to mobility and variances in channel capacity and imposes no control message overhead on the network.**

I. Introduction

The fast spread of small wireless computers has enabled the design and deployment of wireless ad hoc networks. Typical applications proposed for such networks include both realtime and best effort applications. While realtime applications, such as audio/video or on-demand multimedia retrieval, require quality of service (QoS) guarantees for effective communication, best effort applications, such as file transfer, are more tolerant to changes in bandwidth and delay. To support both types of applications in ad hoc networks, effective QoS-aware protocols must be used to allocate resources to flows and provide guarantees to realtime traffic in the presence of best effort traffic.

The unique characteristics of ad hoc networks impose great challenges on the design of such QoS-aware protocols. First, since an ad hoc network has no centralized control, only local information is available to any node in the network. Therefore, QoS-aware protocols for ad hoc networks must use distributed algorithms and not rely on global information. In addition, the shared nature of the wireless channel makes resource allocation very complex since allocation of resources at an individual node affects available resources at its contending neighbors, which may be outside of its communication range. Furthermore, the mobility of nodes may often break connections or resource reservations at each node, incurring high message overhead from the need to reconfigure the reservations. Finally, the wireless channel is highly unreliable and its capacity may vary dramatically. Therefore, QoS-aware protocols should not be sensitive to packet loss or rely on exact knowledge of channel capacity.

The design of QoS-aware protocols under these challenges is non-trivial and requires coordination between different layers of the protocol stack. The goal of our research is to provide a cross-layer QoS mechanism, QPART (**Q**oS **P**rotocol for **A**dhoc **R**ealtime **T**raffic), that can support QoS for realtime traffic under these challenges. Contrary to existing approaches, QPART does not depend on control message exchanges between contending neighbors to coordinate resource allocations. Instead, QPART achieves QoS-aware resource allocation by dynamically adapting the contention window sizes of nodes based on local network congestion levels and received QoS. Since no explicit control messages are used, QPART imposes no message overhead and the loss of control messages does not affect its operation. In addition, QPART does not require a node to keep any static QoS state, eliminating the need for expensive reconfiguration in the presence of mobility or changes in channel capacity.

The remainder of this paper is organized as follows. Section II discusses the necessary components for providing QoS support in ad hoc networks and review existing approaches and their limitations in providing the functionality of these components. In Section III, we briefly review the cross-layer architecture of QPART, while Section IV explores the design details of the scheduling part and Section V describes the QoS management part of QPART. Section VI presents our performance evaluation of QPART using simulations. Section VII concludes our work and discusses future research directions.

II. QoS Support in Ad Hoc Networks

Due to the lack of centralized control in an ad hoc network, distributed resource allocation must be used to allocate resources along the routes of flows to provide

flow-based QoS guarantees. Since an ad hoc network has no fixed infrastructure, every node in the network must participate in distributed resource allocation and hence must be equipped with QoS support, which requires three necessary components: admission control, QoS-aware scheduling and conflict resolution. In this section, we identify the functions of these three components and their implementation challenges. We also briefly discuss existing approaches for implementing these components and their limitations in coping with the challenges.

A. Necessary Components for QoS Support

Providing QoS guarantees in an ad hoc network requires three major components at every node in the network. The first component is admission control to ensure that the total resource requirements of admitted flows can be handled by the network. If there are not enough resources for all realtime flows, some realtime flows must be rejected to maintain the guarantees made to other realtime flows. The second component is QoS-aware scheduling, which allocates resources to admitted realtime flows and guarantees their QoS under the condition that admission control is properly performed. The QoS-aware scheduling also regulates the sending rate of best effort flows to prevent them from degrading the QoS of realtime flows. The third component is conflict resolution, which deals with QoS violations and selects victim flows to be rejected to maintain the QoS of the remaining flows. The unique characteristics of ad hoc networks impose three major challenges for designing these three components.

The first challenge is due to the shared nature of the wireless medium. In a wireless network, transmissions from a node not only use local resources, but also consume the bandwidth of neighbors in contention range. Therefore, resource allocation must consider not only the achievable service level of a flow, but also the impact of a flow on the neighboring flows and their available resources [18], greatly enhancing the complexity of resource allocation. Additionally, for many widely available protocols, including IEEE 802.11 [17], carrier sensing is used to provide efficient collision and interference avoidance. In these protocols, the contention range of a node equals its carrier-sensing range, which often is more than twice the transmission range. Therefore, two nodes that consume each other's bandwidth may not be able to decode each other's messages if they are located outside transmission range but inside carrier-sensing range. Coordination between such nodes is non-trivial since their messages must be transmitted either over multiple hops or with higher power, both of which can be very expensive in terms of message overhead [18].

The second challenge is caused by mobility. A broken link causes all flows that traverse this link to be rerouted, requiring new admission control. Therefore, in a mobile network, admission control protocols with high message overhead are highly undesirable due to frequent link breaks in the network. Furthermore, two flows that originally have enough bandwidth may pass through nodes that move into each other's contention range, resulting in degraded service to both flows. To reestablish QoS commitments, one of the flows must be picked as a victim by terminating, rerouting or reducing its QoS requirements. Conflict resolution components for both flows must select the victim based on the priorities of flows and should not result in punishment of both flows. Since the two nodes may be located outside transmission range but inside carrier-sensing range, coordination between the conflict resolution components may be difficult and may have high message overhead if explicit message exchange between conflict resolution components is used.

The third challenge for QoS support is the unreliable and dynamic nature of the wireless channel. Due to fading and outside interference, the wireless channel has a high packet loss rate and the capacity of the channel may change dramatically. In addition, today's wireless devices are able to adapt their coding rates according to channel quality, which may further increase variations in channel capacity. Such dynamics of the channel may compromise QoS protocols that depend on reliable message exchanges between nodes, as well as QoS protocols that rely on explicit knowledge of channel bandwidth. Therefore, all three QoS components must be robust to packet losses and no assumptions about channel capacity should be made.

B. Existing Approaches

Due to these tough challenges, none of the existing QoS protocols for ad hoc networks provide satisfactory solutions for providing all three major components. The existing approaches can be classified into four types, each with its own limitations.

The first type of protocol [4], [7], [12] is designed for a TDMA-based MAC layer. QoS-aware scheduling is achieved by reserving dedicated time slots for realtime flows according to their service requirements. Admission control is performed by looking for a sequence of free time slots, while ensuring that nodes in each other's contention range are allocated with different time slots to avoid collisions. However, a TDMA-based MAC layer requires effective time synchronization between all nodes in the network. Applying highly synchronized solutions in an ad hoc network is expensive and synchronization

can fail when the nodes are mobile. In addition, to ensure that nodes in each other's contention range transmit in different time slots, admission control algorithms at each node must coordinate slot allocation tables with contending neighbors, which is expensive in terms of message overhead. Finally, the slot allocations are very vulnerable to mobility or channel capacity variation since reconfiguration is necessary if two nodes move into each other's contention range or if the coding rate of a channel changes. Hence, conflict resolution must often reconfigure slot allocations, which results in high message overhead and is error prone in the presence of packet losses.

The second type of protocol focuses on QoS-aware scheduling [9], [13] and avoids the cost of time synchronization in TDMA-based MAC layers by operating on a single channel MAC layer, such as IEEE 802.11. In these protocols, each node constructs a neighborhood scheduling table by learning the packet deadline information at its neighbors, which is piggy-backed in RTS-CTS-DATA-ACK handshakes. QoS-aware scheduling can be realized according to the neighborhood scheduling tables. These protocols, however, have high message overhead due to the extra piggy-backed information in the handshake messages. Furthermore, since in a wireless network the area of the carrier-sensing range is much larger than the area of the transmission range, a node can only learn the schedules of a small portion of its contending neighbors from listening to the handshake messages, which greatly affects the effectiveness of these scheduling protocols.

The third type of protocol, including IEEE 802.11e [15] and [1], achieves service differentiation while avoiding the overhead associated with explicit message exchange. In these schemes, flows are grouped into several classes. Service differentiation is achieved by assigning different classes with different contention related parameters such as contention window size, frame size and interframe space. Even though these protocols may guarantee that some class of traffic has *better* service quality than the others, there is no guarantee about whether a flow can get its desired service level. Hence, these protocols do not achieve the goal of QoS-aware scheduling.

The fourth type of protocol, including SWAN [2], VMAC [3], INSIGNIA [10], MMWN [16] and [14], focuses on the admission control and conflict resolution components. These protocols provide admission control through signaling protocols that rely on achievable service level prediction and message exchange along the routes of flows. The local achievable service level at each node is predicted through passive monitoring of the channel. However, these protocols do not give enough attention to the fact that transmissions at one node may reduce the available bandwidth of all nodes in its carrier-sensing range. As a result, these protocols only ensure that a newly added flow achieves its desired QoS but can not prevent the QoS of existing flows from degrading due to the contention of the new flow. A remedy for these protocols is to involve the contending neighborhood in the estimation of achievable service levels by explicit message exchange with these neighbors [18]. However, this remedy has expensive message overhead since some of the contending neighbors may not be located in transmission range and can only be reached through multihop messages. Furthermore, due to high packet loss rates in ad hoc networks, using message exchange to coordinate resource allocation is not reliable or accurate. The varying channel capacity may also invalidate the achievable service level prediction. Finally, since link breaks and QoS violations may be frequent due to mobility, using signaling to reestablish QoS guarantees and perform conflict resolution imposes high message overhead since a new admission control process must be invoked for every link break.

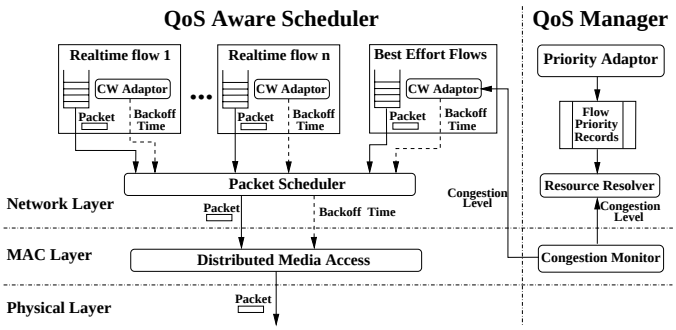

Fig. 1. QPART's architecture

The above discussion illustrates that a QoS protocol should impose minimum or no message overhead and rely on a simple MAC layer. It must also be aware of the effects of local resource allocation on neighborhood nodes. The design of QPART incorporates all of these considerations to effectively support QoS in ad hoc networks.

III. Architecture Overview

To implement the three necessary components for QoS support, QPART consists of two parts, the *QoS-aware Scheduler* and the *QoS Manager*, as shown in Figure 1. Both parts span both the network and the MAC layer. The QoS-aware scheduler realizes the functionality of QoS-aware scheduling, while the QoS Manager implements admission control and conflict resolution. Both parts operate independently and require no message exchanges between neighboring nodes or knowledge of channel bandwidth.

The QoS-aware scheduler is based on an enhanced IEEE 802.11 MAC layer protocol. It exploits the fact

that contention related parameters, specifically contention window size, affect the service quality that a flow receives. Through adaptation of contention window size based on current service quality and network congestion level, the QoS-aware Scheduler guarantees that admitted realtime flows receive their desired services and controls the rate of best effort flows so that they fill the bandwidth left by realtime flows. This contention window adaptation mechanism consists of per-flow queues in the network layer, each with a *Contention Window Adaptor* and a *Packet Scheduler*, which schedules packets from the queues and sends them to the MAC layer (see Section IV).

The QoS Manager performs admission control and conflict resolution based on the priorities of realtime flows and the congestion level of the channel. When the network is congested, the QoS manager picks low priority realtime flows to be rejected. The priorities of realtime flows, which are dynamically assigned by the *Priority Adaptor*, are maintained by the QoS Manager in the *Flow Priority Record*. The congestion level of the channel is fed back from the *Congestion Monitor* in the MAC layer to the *Resource Resolver*, which is responsible for picking victim flows to be rejected based on the channel congestion level and the priority information of flows in the *Flow Priority Record* (see details in Section V). In the next two sections, we present the details of the QoS-aware Scheduler and the QoS Manager.

IV. Distributed QoS-aware Scheduler

The distributed QoS-aware Scheduler of QPART guarantees the QoS of admitted realtime flows under the condition that the capacity of the network is larger than the requirements of all admitted flows. It consists of an underlying MAC protocol and a network layer Packet Scheduler. The design of the QoS-aware Scheduler is based on IEEE 802.11, since it is simple, robust, does not require any centralized control and is a mature technology that has been used in many widely available commercial products. This section explores the design of the QoS-aware Scheduler of QPART and shows how it can guarantee that the admitted realtime flows achieve their required QoS.

A. Distributed Medium Access Control

The MAC algorithm in QPART is based on IEEE 802.11 DCF mode with simple modifications. In this section, we briefly review IEEE 802.11 DCF mode and show how service differentiations can be achieved.

1) IEEE 802.11 DCF Mode: In IEEE 802.11 DCF mode, the transmission of each unicast packet invokes an RTS-CTS-DATA-ACK or DATA-ACK handshake between the sender and the receiver as seen in Figure 2. A

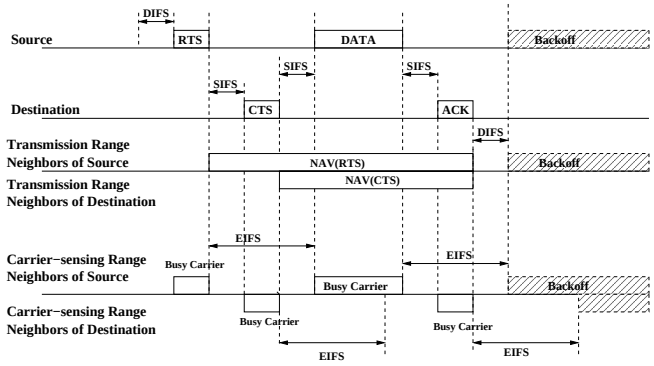

Fig. 2. RTS-CTS-DATA-ACK handshake.

node desiring to transfer a data packet first invokes the carrier-sense mechanism to determine the busy/idle state of the medium based on the *Carrier-sensing Threshold* or a RTS or CTS packet indicating active communication in its neighborhood. If the medium is idle, the node defers a *DCF interframe space* (DIFS). If the medium stays idle during this DIFS period, the node may transmit its RTS packet. If the medium is busy, the node waits until the medium is determined to be idle for DIFS time units if the last detected frame was received correctly or *extended interframe space* (EIFS) time units if the last detected frame was not received correctly. After this DIFS or EIFS idle time, the node defers for an additional backoff period before transmitting an RTS. If the backoff timer is not yet set, the backoff period is generated as *BackoffTime = Random() $\times$ aSlotTime*, where *Random()* is a pseudo random number uniformly distributed between 0 and *contention window (CW)* and *aSlotTime* is a very small time period (20 μs in the IEEE 802.11b standard). The backoff time is decremented by *aSlotTime* if the channel is idle during this period and stopped when a transmission is detected on the channel. The backoff timer is reactivated when the channel is sensed idle again for more than DIFS time units. The node transmits when the backoff timer reaches zero. After each failed transmission attempt, the contention window size is doubled to avoid congestion.

2) Service Quality Differentiation: Since during contention for the channel, the node with the smallest backoff time always wins, the backoff process provides a distributed method to differentiate the service that a node receives. By decreasing the contention window size, a node essentially decreases its average backoff time and hence increases the chances that it wins the channel when competing with other nodes, affecting the node's service quality in terms of bandwidth and packet delay. In [11], contending nodes' contention window sizes and service have been shown to have the following proportional relation-

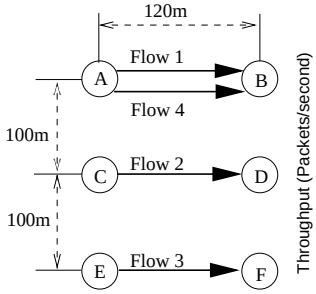

Fig. 3. Topology

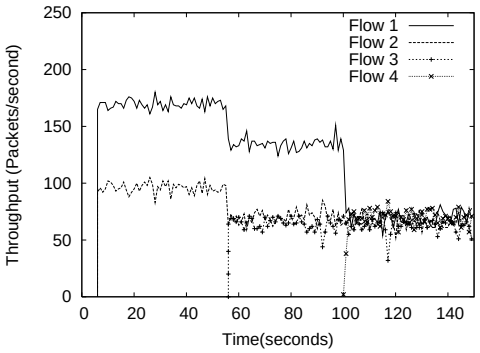

Fig. 4. Throughput of flows

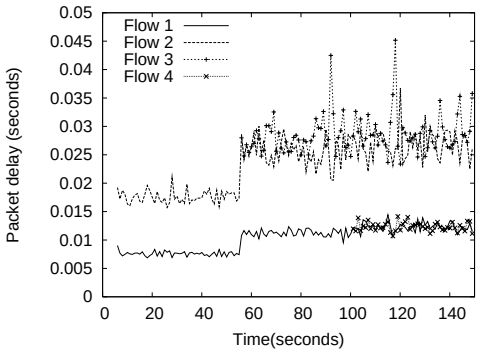

Fig. 5. Delay of flows

ships:

$$\frac{S_i}{S_j} = \frac{\frac{L_i}{CW_i}}{\frac{L_j}{CW_j}}, \tag{1}$$

$$\frac{D_i}{D_j} = \frac{CW_i}{CW_j}, \tag{2}$$

where S_i is the throughput of Node i, D_i is the average packet delay of Node i and L_i is the average packet length of Node i. Based on this proportional relationship between contending nodes' contention window sizes and services in terms of bandwidth and delay [11], IEEE 802.11e [15] and [1] allocate different contention window sizes to different classes of traffic so that class-based proportional fairness can be achieved. However, proportional fairness provides no guarantees about the actual service quality that a flow receives. As the number of competing nodes and flows increases, the actual service quality to every flows decreases.

To demonstrate this, we run a simple simulation using the NS2 simulator [5]. The topology of the simulation is shown in Figure 3, where Flows 1 and 4 belong to the same class with contention window size 31 and share the same route. Flows 2 and 3 belong to another traffic class with contention window size 63. The capacity of the channel is around 270 packets per second. Flows 1 and 2 start at the 5th second, Flow 3 starts at the 55th second and Flow 4 starts at the 100th second. The throughput and delay of all flows are shown in Figures 4 and 5. At the beginning, Flow 1 gets 2/3 of the bandwidth and Flow 2 gets 1/3 of the bandwidth. The delay of Flow 1 is half of the delay of Flow 2. After Flow 3 starts, the throughput of Flow 1 decreases to half of the bandwidth and the throughput of Flows 2 and 3 are each 1/4 of the bandwidth. The delay of both Flows 1 and 2 increase. After Flow 4 starts, the throughput of Flows 1 and 4 become 1/4 of the bandwidth while the throughput of Flows 2 and 3 remains unchanged. This example shows that as the number of competing nodes and flows changes, the actual service quality to flows changes dramatically. It demonstrates that static allocation of contention window sizes can not provide any guarantees to the service qualities of flows.

B. Dynamic Contention Window Adaptation

Since static contention window allocation only provides proportional differentiation, dynamic contention window adjustment can be used to adapt the contention window sizes according to the network environment and the requirements of flows. Therefore, each realtime flow in QPART has its own packet queue and contention window and accesses the channel as if it is an independent node. In this section, we explain how the contention window size of a realtime flow is adjusted by the Contention Window Adaptor in QPART so that the service quality received by a flow can meet the flow's service requirement.

Since flows may have different QoS requirements based on the type of application data carried in the flow, QPART classifies flows into three types: delay-sensitive flows, bandwidth-sensitive flows and best effort flows. The delay-sensitive flows, such as conversational audio/video conferencing, require that packets arrive at the destination within a certain delay bound. The bandwidth-sensitive flows, such as on-demand multimedia retrieval, require a certain throughput. The best effort flows, such as file transfer, can adapt to changes in bandwidth and delay. Due to the different requirements of flows, each type of flows has its own contention window adaptation rule.

1) Delay-Sensitive Flows: For a delay-sensitive flow, the dominant QoS requirement is end-to-end packet delay. To control delay, the end-to-end delay requirement d must be broken down into per-hop delay requirements. Each hop locally limits packet delay below its per-hop requirement to maintain the aggregated end-to-end delay below d. For this paper, each node is assigned with the same per-hop delay requirement, d/m, where m is the hop count of the flow. It is also possible to allocate per-hop delay requirements based on node traffic load or the packet's

achieved service. We are currently investigating the effects of different per-hop delay allocation schemes.

To set up the contention window adaptation, the first few packets of the flow piggyback the per-hop delay requirement to relying nodes. At each relaying node, the Contention Window Adaptor adapts the contention window to ensure per-hop packet delay requirements of the flow. If the delay of the flow is too large, the adaptation algorithm at the node decreases the contention window size of the flow to decrease the delay. On the other hand, if the delay is smaller than the requirement, the adaptation algorithm increases the contention window size so that the channel resources can be left for use by other flows. Based on this approach, every node along the route of the flow periodically updates the flow's contention window CW according to the following iterative algorithm:

$$CW^{(n+1)} = CW^{(n)} \times (1 + \alpha \frac{d/m - D^{(n)}}{d/m}), \quad (3)$$

where the superscript n represents the n^{th} update iteration, D denotes the actual peak packet delay at the node during a update period and α is a small positive constant. Essentially, this iterative algorithm adjusts the contention window size of the flow to maintain the packet delay below its per-hop delay requirement. The step size α and the update interval should be picked appropriately to produce fast response to changes in network condition. In our simulation, the update period is 100 ms and the α is set as 0.1. The settings of all the QPART parameters used in our simulations can be found in Table I in Section VI.

2) Bandwidth-Sensitive Flows: For a bandwidth-sensitive flow, the dominant QoS requirement is throughput, which requires that at each node along the flow's route, the packet arrival rate of the flow should match the packet departure rate of the flow. According to queueing theory, the flow's queue length should be finite. Therefore, by maintaining a constant queue length, the throughput of the flow can be guaranteed. Hence, the contention window adaptor for a bandwidth-sensitive flow updates the contention window periodically as follows:

$$CW^{(n+1)} = CW^{(n)} + \beta(q - Q^{(n)}), \quad (4)$$

where q is a threshold value of the queue length that is smaller than the maximum capacity of the queue, Q represents the actual queue length and β is a positive constant. If Q is larger than q, the algorithm decreases CW to increase the packet departure rate to decrease queue length. If Q is smaller than q, the algorithm increases CW to decrease the packet departure rate and free up resources for other flows. As the queue size varies around the threshold value q, the average throughput of the flow matches its requirement. The threshold size q should be much smaller than the capacity of the queue so that a burst of traffic does not cause packet loss due to queue overflow. Currently we set q according to the guidelines provided in the popular queue management protocol RED [6].

3) Best Effort Flows: Best effort flows are tolerant to changes in service levels and do not have any hard requirements about bandwidth or packet delay. Since there is no per-flow service requirement, all packets from best effort flows are put in a common queue and only one contention window parameter is kept for all best effort flows. The purpose of updating the contention window size of best effort flows is to prevent best effort flows from congesting the network and degrading the service level of realtime flows. To achieve this, the contention window of best effort flows is updated as follows:

$$CW^{(n+1)} = CW^{(n)} \times (1 + \gamma(f - F^{(n)})), \quad (5)$$

where f is a *congestion threshold* for idle channel time, F is the actual idle channel time and γ is a positive constant. Here, *idle channel time* is defined as the average length between two consecutive busy periods of the channel, which decreases as the load on the network increases. Guidelines for setting f are discussed in Section V-C.

The iterative algorithm in Equation (5) updates the contention window size of best effort flows to avoid network congestion. When the average idle channel time F is smaller than the threshold value f, the network is considered congested and the contention window size of the best effort traffic is increased to avoid decreasing the service level of realtime traffic. On the other hand, if the network is lightly loaded so that the idle channel time is larger than f, the contention window size of best effort traffic is decreased so that the idle bandwidth can be utilized.

The design of the above three contention window adaptation algorithms ensures that realtime flows dynamically adjust their contention parameters to meet their own QoS needs. A realtime flow that did not get its required QoS in the past due to competition from other flows decreases its contention window size so that statistically it will have a higher chance to obtain the channel in the future. A best effort flow, on the other hand, increases its contention window size when the network is considered busy and hence releases the channel to the realtime flows. The random generated backoff counter ensures that the channel access attempts from different flows are spread out and do not cause a lot of collision. Contrary to [9], [13], in QPART, no neighborhood scheduling tables and piggybacked schedule information are needed. Therefore, there is no control message overhead imposed by QPART and

the schedules of packets are not affected by channel errors. Since the QoS-aware Scheduler does not require any knowledge of channel capacity, variations in the channel capacity do not affect the performance of QPART.

C. Implementation Considerations

To implement the contention window adaptation algorithm, flows must adapt their contention window sizes and access the channel independently as if they are individual nodes in an IEEE 802.11 network. Otherwise, head-of-line blocking between flows in the same node may limit the effectiveness of the contention window adaptation algorithm. However, implementation of a per-flow contention mechanism at the MAC layer is difficult since the MAC layer has no access to flow information. Additionally, it is not desirable to require the MAC layer to recognize flow types and adapt the contention window sizes based on the QoS of flows. To solve this problem, QPART makes minimum changes to the MAC layer and implements the contention window adaptation algorithm mainly at the network layer, where flow information is accessible. The only modification to the MAC layer is that the MAC layer is relieved from the task of keeping the contention window and calculating the backoff time. Instead, the backoff time is calculated in the network layer where the contention windows of flows are kept and the backoff time is sent to the MAC layer along with data packets.

To simulate the effect that each flow access the channel independently using their own contention window, it is important to understand the contention resolution process. Consider that there are multiple contending flows, each is associated with its own contention window. After the channel turns from busy to idle for DIFS time, every flow starts to count down their backoff timer once per *aSlotTime*. Assume that Flow i has the smallest backoff time so that its backoff timer reaches 0 first. Therefore, Flow i wins the channel and starts transmission. The other flows pause their backoff timers at a value that equals their original backoff time minus the backoff time of Flow i. When the transmission of Flow i stops, Flow i generates the next backoff time and all flows resume their backoff process.

To realize this contention resolution process, at each flow's queue in QPART, the head of line (HOL) packet is associated with a backoff time, which is generated by the *Backoff Generator* as a random number in $[0, CW]$ multiplied by *aSlotTime*, where CW is the contention window size of the flow. At each node, whenever the MAC layer is ready to transmit a packet, the Packet Scheduler selects the HOL packet with the smallest backoff time inside the node and delivers it along with its backoff time to the MAC layer. The backoff times of the remaining HOL packets are reduced by the backoff time of the chosen packet. When the MAC layer receives the chosen HOL packet, it backs off according to the backoff time received along with the packet. The node with the smallest backoff time at the MAC layer wins the channel, which essentially means that the flow with the smallest backoff time among all contending flows wins the channel.

When the wining node finishes the transmission successfully, its MAC layer informs the network layer so that the transmitted packet is removed from its queue and a new HOL packet and its corresponding backoff time can be generated. If the packet transmission of the winning node fails, its MAC layer informs the network layer of the failure so that the failed packet remains as HOL packet in its queue. In this case, the contention window size of the queue is doubled and a new backoff time is regenerated for the retransmission of the failed packet. Finally, after the successful or failed packet transmission, the Packet Scheduler at the node again selects the HOL packet with the smallest backoff time among all queues and delivers it along with its backoff time to the MAC layer for the next transmission.

Figure 6 shows an example of this process. In this example, there are three flows. Flows 1 and 2 are in Node A, while Flow 3 is in Node B. Initially, each flow has two packets in its queue, where $P_{i,j}$ represents the i_{th} packet in Flow j's queue. Flow 1's HOL packet $P_{1,1}$ is associated with the smallest backoff time 5 among all the flows in Node A. Hence, the packet scheduler delivers $P_{1,1}$ to the MAC layer along with the backoff time 5. The backoff time of $P_{1,2}$, which is the HOL packet of Flow 2's queue, is updated from 7 to 2 by subtracting the backoff counter of $P_{1,1}$. Similarly, the Packet Scheduler in Node B delivers packet $P_{1,3}$ along with backoff time 7 to its MAC layer. After backing off 5 *aSlotTime*, the MAC layer in Node A wins the contention and transmits $P_{1,1}$ and the backoff time of Node B stops at 2. After the transmission of $P_{1,1}$, $P_{1,1}$ is removed from Flow 1's queue and $P_{2,1}$ becomes the new HOL packet in Flow 1's queue. A new backoff counter 9 is generated for packet $P_{2,1}$. At this time, the HOL packet with the smallest backoff time in Node A is $P_{1,2}$. Therefore, $P_{1,2}$ is chosen by the Packet Scheduler and is sent to the MAC layer of Node A for transmission. At this time, the MAC layer of Node B wins the channel since it has the smallest backoff time at the MAC layer and packet $P_{1,3}$ is transmitted.

The above example shows that with minimum changes to the MAC layer, each flow in QPART's per-flow contention mechanism acts like an independent node in an IEEE 802.11 network. There is no head-of-line blocking between flows in the same node. Each flow updates their

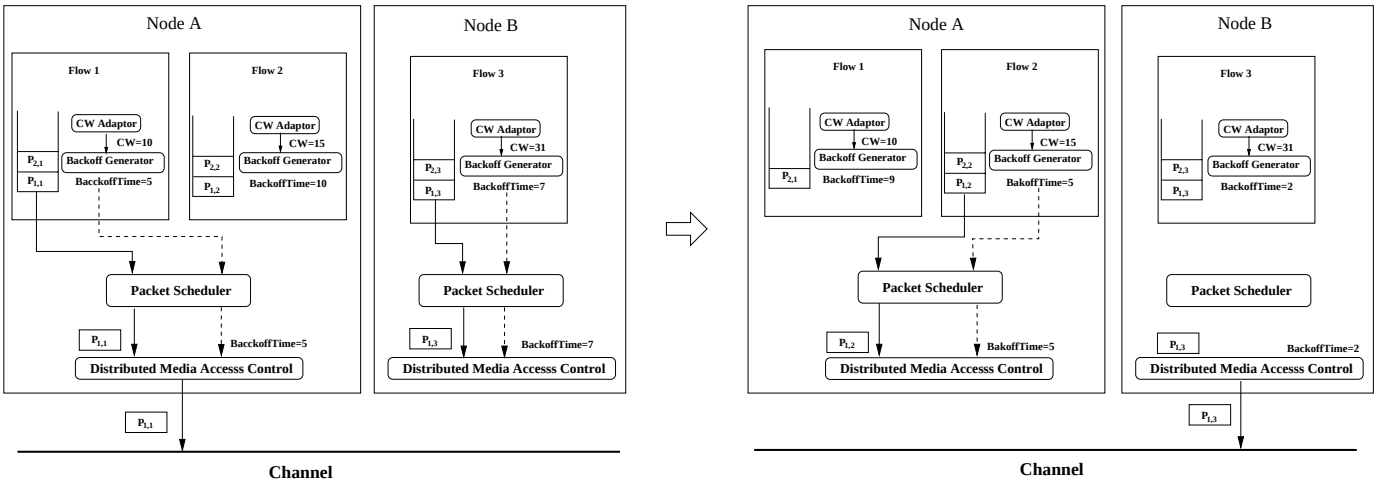

Fig. 6. QoS-aware Packet Scheduler

contention window independently and controls their own channel access frequency. This independence ensures that the QoS guarantees can be achieved by adapting the contention window sizes of the flows.

V. QoS Manager

The goal of the QoS manager is to realize admission control and conflict resolution by maintaining the total requirements of realtime traffic below network capacity. When the total requirements exceed network capacity, the QoS Manager selects victim flows and informs their senders to either terminate, reroute or reduce QoS requirements. The selection process of victim flows is based on the prioirties of flows, which are dynamically assigned based on network policies. In this section, we first discuss the design of the two components of QPART, the *Priority Adaptor* and the *Resource Resolver*, and then analyze the relationship between the contention window adaptation algorithm of best effort traffic and the QoS Manager.

A. Priority Adaptor

The QoS Manager selects victim flows based on the priorities of flows, which are assigned to flows according to network policies. The actual choice of a priority assignment algorithm is orthogonal to the design of QPART, which simply enforces the priorities once they are chosen. In this section, we describe several possible options for priority allocation policies, which can be used solely or combined together to satisfy different network requirements.

The first policy allocates priorities to flows according to their importance. The more important a flow is, the higher its priority should be. Using this policy, a newly arrived flow with higher importance can kick flows with lower importance out if the network does not have enough resources for all flows. For ad hoc networks that are used in emergency situations, this allocation policy is appropriate since messages with high importance should be able to kick out unimportant messages if needed.

For an ad hoc network that does not have classified importance level, a policy that allocates priorities to flows according to their age may be desirable, which have been implemented in our simulations. With this policy, the Priority Adaptor increases the priority, P, of a flow k periodically as follows:

$$\begin{cases} P_k^{(n+1)} = P_k^{(n)} + 1, & \text{if} P_k^{(n)} < P_{max}, \\ P_k^{(n+1)} = P_k^{(n)}, & \text{if} P_k^{(n)} = P_{max}, \end{cases} \quad (6)$$

where P_{max} is the highest possible priority. This priority adpatation algorithm essentially allocates higher priorities to existing flows so that the newly arrived flows are more likely to be rejected than existing flows.

It is also possible that some network may need to penalize realtime flows that exist too long since these flows consume too much bandwidth and reduce the capacity of network that can be used by other flows. The policy of this network is to periodically decrease the priority of a flow if the age of the flow is longer than a certain threshold. It is also possible to increase the number of simultaneous flows by discouraging high QoS requirement flows. In this case, a flow requiring a high service level is allocated with a lower priority since it requires more resources from the network and potentially reduces the number of flows the network can sustain.

Different priority adaptation rules may also be combined together to provide a variety of admission policies. For example, the age-based policy may be combined with the importance-based priority policy to give priorities between flows with the same level of importance. The

choice of which adaptation rules to use depends on the needs of the ad hoc network. We are currently investigating the use of different priority policies and their impoact on QoS in wireless networks.

B. Resource Resolver

The QoS-aware Scheduler in Section IV-B guarantees that admitted realtime flows achieve their desired QoS when network capacity is larger than the requirements of the realtime traffic. However, if the total admitted realtime flows exceed the capacity of the network, no scheduling algorithm can guarantee the QoS of flows. The goal of the Resource Resolver is to ensure that the total requirements of realtime flows are smaller than network capacity through admission control and conflict resolution.

In wired networks, a node has global knowledge of the flows sharing its link and centralized control of its link bandwidth allocation. Therefore, a node can easily accurately predict a new flow's impact on existing flows and the new flow's expected service. Therefore, admission control in wired networks is traditionally performed before the new flow starts. If the new flow passes admission control, it is guaranteed that there are always enough resources for the flow and conflict resolution is not needed.

On the contrary, a wireless node has no centralized control of its bandwidth allocation or global knowledge about the flows that are competing for the channel. Therefore before the new flow starts, it is very difficult to accurately predict the impact of a new flow on the service of existing flows or the expected service of the new flow. In addition, even if the new flow passes admission control, there is no guarantee that there are always enough resources for the flow due to the mobility of nodes and variances in the channel capacity. Therefore, we argue that there is little value and high cost to perform admission control before a new flow starts. Instead, admission control should only be activated to reject flows when the new flow actually affects the service of existing flows or cannot get its desired service. However, conflict resolution is a necessity due to the dynamics of the wireless network.

Based on this rationale, we design the Resource Resolver of QPART to realize admission control and conflict resolution following an on-demand methodology. When a new flow arrives, it starts transmitting packets immediately. The QoS-aware Scheduler guarantees that if there are enough resources, the existing realtime flows can adapt their contention windows to maintain their desired QoS. When there is not enough network capacity, the Resource Resolver detects the increased congestion level in the network on demand and appropriately selects victim flows to be rejected to maintain the QoS of the remaining flows. The Resource Resolver can similarly handle resource shortages caused by active flows moving into each other's contention range or decreasing channel capacity. Essentially, the Resource Resolver realizes the goals of both admission control and conflict resolution by preventing network congestion caused by newly arrived flows, mobility of nodes or variances in channel capacity.

To understand how the Resource Resolver detects increased congestion, note that the QoS-aware Scheduler adapts the contention window sizes of realtime flows to try to satisfy the QoS of the flows. If there are not enough resources, none of the realtime flows can achieve their desired QoS and they repeatedly decrease their contention window sizes down to zero. In this case, the packet collision rate is so high that the network throughput decreases to zero. Such a contention window "blow out" signals a resources shortage for realtime flows. Therefore, QPART runs a Congestion Monitor at the MAC layer, which passively monitors idle channel time. When the contention window sizes of flows start to decrease due to resource shortages, the idle channel time decreases, triggering victim flow selection by the Resource Resolver to prevent contention window blow out.

To select victim flows, the priority of a flow is mapped to a threshold value of the idle channel time, called the *admission threshold*. The higher the priority, the smaller the admission threshold. Let P_k be the priority of flow k and θ be the difference between the admission thresholds of two consecutive priority levels. The admission threshold, T, for flow k can be expressed as:

$$T_k = (P_{max} - P_k) \times \theta + \eta, \tag{7}$$

where η is the smallest admission threshold. When idle channel time goes below the admission threshold of a flow, a flow becomes a rejection candidate in the Resource Resolver. Rejection of the candidate flows reduces congestion on the channel and increases channel idle time. Therefore, contention window blow out is avoided. To prevent several flows with the same priority level from being rejected simultaneously and to eliminate the effects of temporary interference from outside sources, before a rejection candidate flow k gets rejected, the Resource Resolver waits a random short period of time, called the *rejection defer time*, which is randomly generated in the range $[t_1, t_2]$. t_1 and t_2 are priority related bounds of the rejection defer time and are calculated as:

$$\begin{aligned} t_1 &= P_k \times \delta, \\ t_2 &= (P_k + 1) \times \delta, \end{aligned} \tag{8}$$

where δ is the interval between t_1 and t_2.

At the end of this rejection defer time, the Resource Resolver rechecks the idle channel time. If the idle channel time is still smaller than the flow's admission threshold, the rejection process starts. The flow's packets are dropped and its sender is informed to terminate, reroute or decrease the QoS requirements of the flow. If at the end of the rejection defer time, the idle channel time is larger than the flow's admission threshold, indicating that congestion has been alleviated due to the rejections of other flows or the absence of the interference, the flow is not rejected and is removed from the rejection candidate list.

Since a higher priority flow always has a smaller admission threshold than a lower priority flow, a lower priority flow always hits its admission threshold earlier. In addition, a higher priority flow always has a longer rejection defer time than a lower priority flow since a higher priority flow's t_1 is larger than the t_2 of a lower priority flow. By setting a large enough θ and δ, the differentiation in the admission thresholds and rejection defer times can ensure that a lower priority flow is always rejected before a higher priority flow. However, too large a δ may reduce the congestion response speed of the Resource Resolver and too large a θ may result in the rejection of flows when the network is not congested. We are currently investigating the tradeoffs for properly setting θ and δ.

When the Resource Resolver rejects a flow, the rejected flow releases the channel resources so that the congestion level of the network decreases and the idle channel time increases. When enough flows are rejected so that the network capacity can accommodate the remaining flows, the idle channel time returns back to a normal level. At this point, the Resource Resolver stops rejecting flows.

Since the Resource Resolvers are completely distributed and require no control message exchanges between neighboring nodes, there is no need to worry about the effects of control message loss. There is also no need for resource reservations since the QoS-aware Scheduler guarantees the service quality to realtime flows. When a flow is rerouted due to link breaks, the unused resources on the old route are immediately released and no update of the reservation information is needed.

C. Contention Window Adaptation Algorithm of Best Effort Flows: Revisited

Since a realtime flow is penalized when the congestion level of the network reaches its admission threshold, the adaptation algorithm of best effort flows must be carefully designed so that they do not cause rejections to any realtime flows. As described in Section IV-B, best effort flows increase their contention window sizes when the idle channel time is smaller than the congestion threshold to decrease the congestion level of the network. To guarantee that contention from best effort flows does not decrease the idle channel time below the admission thresholds of realtime flows, the congestion threshold of best effort flows should always be higher than the maximum admission threshold of realtime flows, $P_{max} \times \theta + \eta$. Therefore, as load increases, the best effort flows are the first to reduce their rate before any realtime flow is rejected.

CW update interval			0.1s	Priority update interval			0.1s
α	0.1	β	1	γ	0.1	q	5 pkts
f	1ms	θ	$2\mu s$	η	0.1ms	δ	2ms
P_{max}	250						

TABLE I
CONFIGURATION OF QPART PARAMETERS

VI. EVALUATION

To evaluate the effectiveness of QPART's QoS support, we compare the performance of QPART with SWAN [2] using the NS2 simulator [5]. We choose SWAN since it is the only existing QoS protocol that does not require resource reservation state in the network and claims to achieve low message overhead, which is essentially the same goal as QPART. The NS2 implementation of SWAN is the latest distribution by the SWAN project. The QPART implementation uses the age-based priority policy, where existing flows have higher priorities than new flows. The routing protocol used in the simulations is DSR [8]. The channel bandwidth is 11Mbps. The evaluation demonstrates the performance of QPART in terms of its ability to provide QoS-aware scheduling based on the types of flows and its ability to maintain both delay and bandwidth guarantees to flows in both single and multihop networks. The configuration of QPART parameters are shown in Table I.

A. QoS-aware scheduling

To demonstrate QPART's ability of schedule packets according to QoS requirements of flows, we simulate two five-hop flows competing with each other for bandwidth as shown in Figure 7. Flow 1 starts at time 20s and is delay-sensitive with a delay requirement of 20ms. Flow 2 starts at time 55s and is bandwidth-sensitive. The rates of Flow 1 and Flow 2 are both 30 512Byte packets per second. Figures 8 and 9 show the delay in log scale and the throughput of the two flows. The delay bound of Flow 1 is indicated by the solid line in Figure 8. It can be seen that QPART successfully schedules Flow 1 and Flow 2

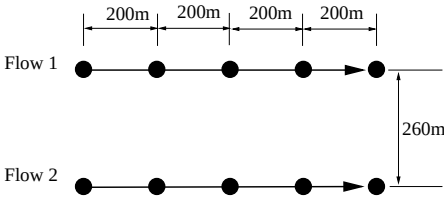

Fig. 7. Topology

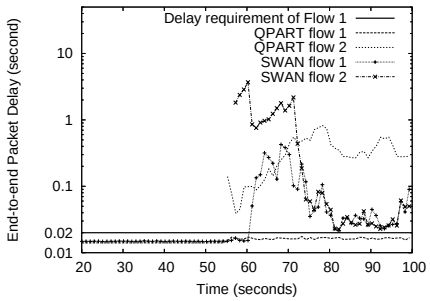

Fig. 8. End-to-end Packet Delay

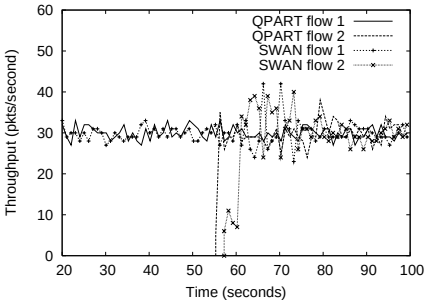

Fig. 9. Flow Throughput

according to their different requirements. QPART maintains the delay of Flow 1 constantly below its delay requirement at the cost of increasing the delay of Flow 2, which is acceptable since Flow 2 is bandwidth-sensitive and QPART maintains its throughput guarantees (see Figure 9). SWAN, however, does not understand the different service requirements of the flows so that both flows experience large delays after Flow 2 starts. Additionally, SWAN does not maintain a stable throughput for Flow 2 from 55s to 80s due to its slow response to queue length increases.

B. QoS guarantees in single hop networks

In this section, we compare QPART and SWAN's ability to keep QoS guarantees in single hop networks. The simulation area is $500m \times 500m$ square and every flow is one hop. Each simulation runs for 200 seconds.

The first set of simulations examines QPART's ability to keep QoS guarantees to delay-sensitive flows. Flow 1 is delay-sensitive with a delay requirement of 5ms and a rate of 40 512Byte packets per second. At time 1s, Flow 1 starts and then 8 to 32 competing flows are injected into the network. To vary the load on the network, the types of competing flows are varied from bandwidth-sensitive CBR flows to delay-sensitive CBR flows with 10ms delay requirements and then to best effort FTP flows. The rate of the competing CBR flows are varied from 20 pkts/second to 50 pkts/second. Figures 10 and 11 show the log-scale packet delay of Flow 1 under QPART and SWAN respectively, where the legends describe the types and rates (pkts/sec) of Flow 1's competing flows. It can be seen that QPART maintains the delay of Flow 1 below its delay requirement under all circumstances while SWAN violates its delay guarantees to Flow 1 in all scenarios as the load of the network increases.

To examine QPART's ability to keep QoS guarantees to bandwidth-sensitive flows, the settings in the second set of simulations are the same as the first simulation, except that Flow 1 is bandwidth-sensitive. Figures 13 and 14 show the throughput of Flow 1 under QPART and SWAN respectively. It can be seen that QPART maintains the throughput of Flow 1 while SWAN violates its bandwidth guarantees to Flow 1 as the network load increases.

C. QoS guarantees in multihop networks

For our final evaluation, we compare QPART and SWAN's ability to keep QoS guarantees for multihop flows. In the simulation, there are 8 delay-sensitive flows, 8 bandwidth-sensitive flows and 8 best effort FTP flows that try to start consecutively during the first 115 seconds of the simulation. The sources and destinations of the flows are randomly selected from 100 nodes located in a $1000m \times 1000m$ square. The hop counts of flows range from 1 to 7. Each delay sensitive flow has a delay requirement of 100ms and generates 50 80Byte packets per second. Each bandwidth-sensitive flow generates 50 512Byte packets per second. The packet size of FTP flows is 512Byte. Figure 12 shows the average delay of the delay-sensitive flows and Figure 15 shows the violation of bandwidth guarantees to bandwidth-sensitive flows, which is the total throughput of the admitted flows subtracted by the total packet generation rate of these flows. QPART maintains the delay of the admitted delay-sensitive flows below their 100ms delay requirement and shows no violations to the bandwidth guarantees. SWAN, however, admits too many flows so that both the delay and bandwidth of its flows degrade as the load of the network increases. In addition, Figure 12 also shows that SWAN has large peaks in the packet delay. These peaks are due to the DSR route discovery messages that flood the network whenever a new flow is added to the network. QPART's packet delay, however, is not affected by the route discovery messages. This is because the route discovery messages are put at the head of the best effort traffic queue and the QoS-aware Scheduler automatically adapts the contention window sizes of realtime traffic and best effort traffic to guarantee the delay for delay-sensitive flows. Therefore, QPART provides more stable packet delay and throughput than SWAN.

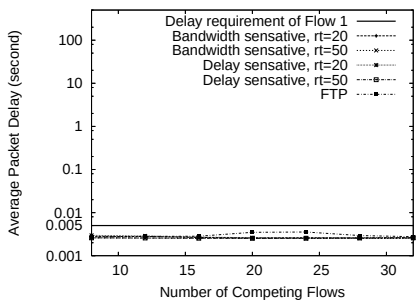

Fig. 10. QPART's delay guarantees to Flow 1 (single hop).

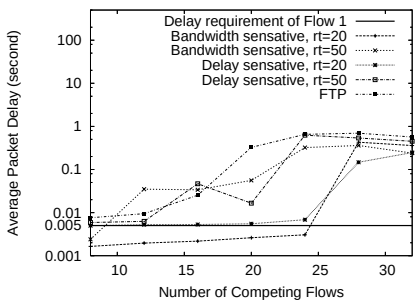

Fig. 11. SWAN's delay guarantees to Flow 1 (single hop).

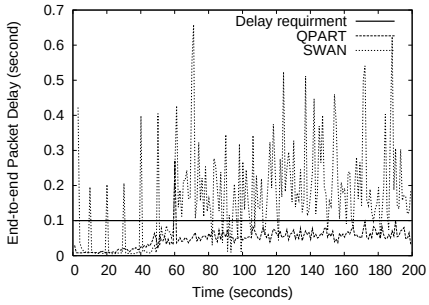

Fig. 12. Delay guarantees to delay-sensitive flows (multihop).

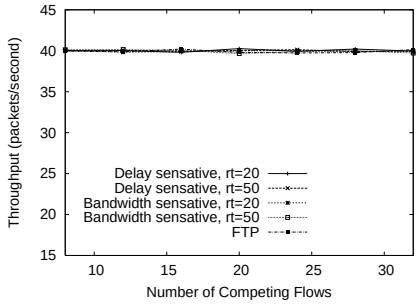

Fig. 13. QPART's bandwidth guarantees to Flow 1 (single hop).

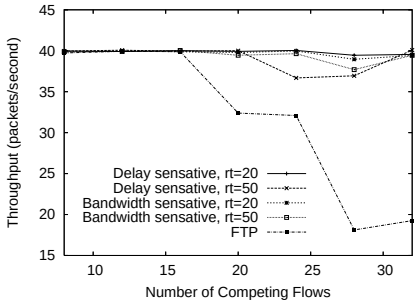

Fig. 14. SWAN's bandwidth guarantees to Flow 1 (single hop).

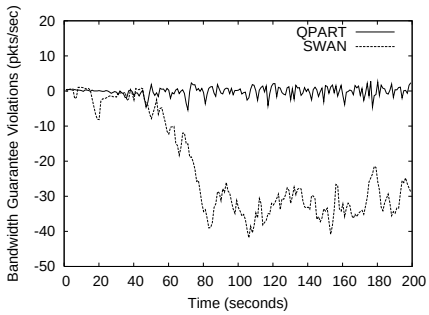

Fig. 15. Bandwidth guarantees to bandwidth-sensitive flows (multihop).

VII. Conclusions

In this paper, we introduce a new QoS support protocol QPART, which is simple, distributed and light weight. QPART support different types of traffic by offering both delay guarantees and bandwidth guarantees. An important benefit of QPART is that it does not require the network to maintain resource reservation states and has very low message overhead since complex signaling is not needed. Through simulations, we compare the performance of QPART with SWAN and demonstrate QPART's ability to provide delay and bandwidth commitments to flows. In the future, we will investigate the effects of different values of the parameters, including q, f, θ and η, on the performance of QPART.

References

[1] Imad Aad and Claude Castelluccia. Differentiation Mechanisms for IEEE 802.11. In *Proceedings of INFOCOM*, 2001.

[2] Gahng-Seop Ahn, Andrew Campbell, Andras Veres, and Li-Hsiang Sun. SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks. In *Proceedings of Infocom*, 2002.

[3] Michael G. Barry, Andrew T. Campbell, and Andras Veres. Distributed Control Algorithms for Service Differentiation in Wireless Packet Networks. In *Proceedings of Infocom*, 2001.

[4] T.-W. Chen, J. T. Tsai, and M. Gerla. QoS Routing Performance in Multihop Multimedia Wireless Networks. In *Proceedings of IEEE International Conference on Universal Personal Communications (ICUPC)*, 1997.

[5] Kevin Fall and Kannan Varadhan. NS notes and documentation. In *The VINT Project, UC Berkely, LBL, USC/ISI, and Xerox PARC*, 1997.

[6] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[7] Y.-C. Hsu and T.-C. Tsai. Bandwidth Routing in Multihop Packet Radio Environment. In *Proceedings of the 3rd International Mobile Computing Workshop*, 1997.

[8] David B Johnson and David A Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[9] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed Multi-Hop Scheduling and Medium Access with Delay and Throughput Constraints. In *Proceedings of the seventh annual international conference on Mobile computing and networking (MobiCom'01), Rome, Italy*, 2001.

[10] Seoung-Bum Lee, Gahng-Seop Ahn, Xiaowei Zhang, and Andrew Campbell. INSIGNIA: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks. *Journal of Parallel and Distributed computing, Special issue on Wireless and Mobile Computing and Communications*, 60:374–406, 2000.

[11] Bo Li and Roberto Battiti. Performance Analysis of An Enhanced IEEE 802.11 Distributed Coordination Function Supporting Service Differentiation. In *International Workshop on Quality of Future Internet Service*, 2003.

[12] C.R. Lin and Chung-Ching Liu. An On-demand QoS Routing Protocol for Mobile Ad Hoc Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, 2000.

[13] Haiyun Luo, Songwu Lu, Vaduvur Bharghavan, Jerry Cheng, and Gary Zhong. A Packet Scheduling Approach to QoS support in Multihop Wireless Networks. *ACM Journal of Mobile Networks and Applications(MONET), Special Issue on QoS in Heteroge-*

neous Wireless Networks, 2002.

[14] D. Maltz. Resource Management in Multi-hop Ad Hoc Networks. In *Technical Report CMU CS 00-150, School of Computer Science, Carnegie Mellon University*, July 2000.

[15] Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, and Lothar Stibor. IEEE 802.11e Wireless LAN for Quality of Service. In *Proceedings of European Wireless*, 2002.

[16] Ram Ramanathan and Martha Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1):101–119, 1998.

[17] IEEE Computer Society. 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[18] Yaling Yang and Robin Kravets. Contention-aware admission control for ad hoc networks. Technical Report UIUCDCS-R-2003-2337, April 2003.